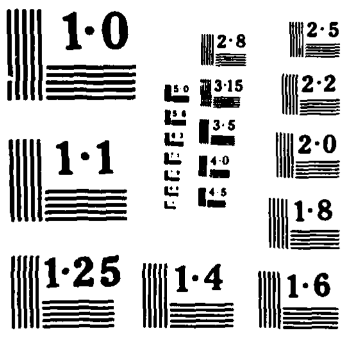
COMPUTATION OF GEOMETRIC PROPERTIES FROM THE MEDIAL AXIS TRANSFORM IN 0 (..(U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH A Y WU ET AL. JUN 85 1/1

SSIFIED CAR-TR-122 AFOSR-TR-85-0616 F/G 12/1 NL



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

CAR-TR-122
CS-TR-1497

F-49620-83-C-0082
June 1985

COMPUTATION OF GEOMETRIC PROPERTIES FROM THE MEDIAL AXIS TRANSFORM IN $O(n \log n)$ TIME

Angela Y. Wu
Department of Mathematics, Statistics, and Computer Science
The American University
Washington, DC 20016

S.K. Bhaskar
Azriel Rosenfeld
Center for Automation Research

COMPUTER VISION LABORATORY

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

A

CAR-TR-122
CS-TR-1497

F-49620-83-C-0082
June 1985

COMPUTATION OF GEOMETRIC PROPERTIES FROM THE MEDIAL AXIS TRANSFORM IN $O(n \log n)$ TIME

Angela Y. Wu
Department of Mathematics,
Statistics, and Computer Science
The American University
Washington, DC 20016

S.K. Bhaskar
Azriel Rosenfeld
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

The digital medial axis transform (MAT) represents an image subset S as the union of maximal upright squares contained in S. Brute-force algorithms for computing geometric properties of S from its MAT require time $O(n^2)$, where n is the number of squares. Over the past few years, however, algorithms have been developed that compute properties for a union of upright rectangles in time $O(n \log n)$, which makes the use of the MAT much more attractive. We review these algorithms and also present efficient algorithms for computing union-of-rectangle representations of derived sets (union, intersection, complement) and for conversion between the union of rectangles and other representations of a subset.

The support of the US Air Force Office of Scientific Research under Contract F-49620-83-C-0082 is gratefully acknowledged, as is the help of Sandra German in preparing this report.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD A158 934

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS: N/A
2a SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): CAR-TR-122 CS-TR-1497	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 85-0616
6a NAME OF PERFORMING ORGANIZATION: University of Maryland; 6b. OFFICE SYMBOL (If applicable): N/A	7a NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c ADDRESS (City, State and ZIP Code): Center for Automation Research College Park, MD 20742	7b ADDRESS (City, State and ZIP Code): Bolling Air Force Base Washington, DC 20332
8a NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-83-C-0082
8c ADDRESS (City, State and ZIP Code): Bldg. 410 Bolling AFB, D.C. 20332-6448	10 SOURCE OF FUNDING NOS.: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. K2; WORK UNIT NO.
11 TITLE (Include Security Classification): Computation of geometric properties from the medial axis transform in O(n logn) time	
12 PERSONAL AUTHOR(S): Angela Y. Wu, S. K. Bhaskar, Azriel Rosenfeld	
13a TYPE OF REPORT: Technical; 13b TIME COVERED: N/A FROM ____ TO ____	14 DATE OF REPORT (Yr., Mo., Day): June 1985; 15 PAGE COUNT: 37
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES: FIELD / GROUP / SUB. GR.	18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The digital medial axis transform (MAT) represents an image subset S as the union of maximal upright squares contained in S. Brute-force algorithms for computing geometric properties of S from its MAT require time O (n^2), where n is the number of squares. Over the past few years, however, algorithms have been developed that compute properties for a union of upright retangles in time O (n logn), which makes the use of the MAT much more attractive. We review these algorithms and also present efficient algorithms for computing union-of-retangle representations of derived sets (union, intersection, complement) and for conversion between the union of rectangles and other representations of a subset.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. Robert N. Buchal	22b TELEPHONE NUMBER (Include Area Code): (202)767-4939; 22c OFFICE SYMBOL: NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

1. Introduction

The medial axis transform (MAT) of a set S was first introduced by Blum [1]. The MAT can be defined as the set of centers and radii of the maximal disks that are contained in S. The "disks" can be of any desired standard shape. For example, if S is a subset of a digital image, it is convenient to use upright squares of odd side length as "disks". The original set is just the union of these locally maximal squares.

In general, the squares overlap and the number of squares, say n, in the MAT of a region is large. If brute-force algorithms are used, using the MAT to calculate geometric properties of a region requires $O(n^2)$ computation, which can be quite large. For this reason, it has been concluded [2] that the MAT is not a very good region representation.

The purpose of this paper is to show that the MAT is not such an unattractive region representation. In the geometric complexity literature over the past few years, many algorithms have been published that compute geometric properties for regions represented as union of upright rectangles in $O(n \log n)$ time. (An upright, "rectilinear", or "iso-oriented" rectangle is a rectangle whose sides are parallel to the coordinate axes.) Clearly, a MAT is a special case of this representation where all the rectangles are squares and have odd side lengths. We will review and discuss algorithms that find the boundary and compute the perimeter, identify the connected components, and find the area and other moments of a union of upright rectangles.

Section 2 introduces the segment tree [3] which is the basic data structure used in the algorithms. Section 3 discusses two algorithms [4,5] that find the boundaries and perimeter of a region. In Section 4, we discuss the connected component algorithm of [6], which uses a priority search tree [7]. We also present an algorithm to solve the connected component problem using the segment tree instead of the priority search tree. Our algorithm has the same time and space complexity as the algorithm in [6]. Section 5 describes an algorithm given in [9] to find the area. We also show how to extend the algorithm of [7] to find the centroid and other moments of the region.

A set can be the result of performing set-theoretic operations on given sets. Section 6 presents algorithms to compute union of rectangle representations for such derived sets, directly from the representations of the given sets. Conversion between unions of rectangles and other representations is discussed in Section 7.

Another approach to speeding up computation based on the MAT is to augment it with additional information. Ahuja and Hoff in [12] introduce an augmented MAT, or AMAT; it makes use of a graph structure in which overlapping squares are joined by arcs. This graph structure takes $O(n \log n)$ time to build. They observe that if the average number of neighbors per square in the AMAT is α, most $O(n^2)$ computations can be performed in $O(\alpha n)$ time. In the reported examples, α varied from 8.3 to 21.9, but this was after pruning the MAT to eliminate many redundant squares. The methods described in this paper take $O(n \log n)$ time, but do not require storage of the graph structure.

2. Segment Trees

The segment tree, introduced in [3], is a useful data structure in many of the algorithms which solve geometric problems involving a union of iso-oriented rectangles using the line sweep method. A segment tree is a special binary tree which allows fast insertions and deletions of line segments when the tree represents line intervals.

Let $[a, b]$ be any interval with $b-a \geq 1$ and for simplicity, let the endpoints be integers. The segment tree $T(a, b)$ is defined as follows: $T(a, b)$ has a root v, with $L(v) = a$ and $R(v) = b$, representing interval $[a, b]$. If $b-a > 1$ then v has a leftson $T(a, \lfloor \frac{a+b}{2} \rfloor)$ and a rightson $T(\lfloor \frac{a+b}{2} \rfloor, b)$. If $b-a = 1$ then leftson(v) = rightson(v) = null. An interval $[c, d] \subseteq [a, b]$ is represented on $T(a, b)$ by a set of marked nodes consisting of the first node v on each of the paths from the root of $T(a, b)$ such that $[L(v), R(v)] \subseteq [c, d]$, i.e., $[L(\text{parent}(v)), R(\text{parent}(v))] \not\subseteq [c, d]$. See Figure 1 for an example. It is clear that for each $[c, d]$, the children of a node cannot both be marked and at each level of the tree there are at most two marked nodes. Since $T[a, b]$ has height $1 + \lceil \log_2 k \rceil$ where $k = b-a+1$, an interval $[c, d]$ is represented by $\leq 2 * (1+\lceil \log_2 k \rceil) = O(\log_2 k)$ nodes. A set of m intervals can be represented in $T[a, b]$ by appending to each tree node the list of intervals which mark the node. Hence a segment tree $T[a, a+k-1]$ with m intervals needs $O(k+m \log_2 k)$ space and $O(k+m \log_2 k)$ time to be built.

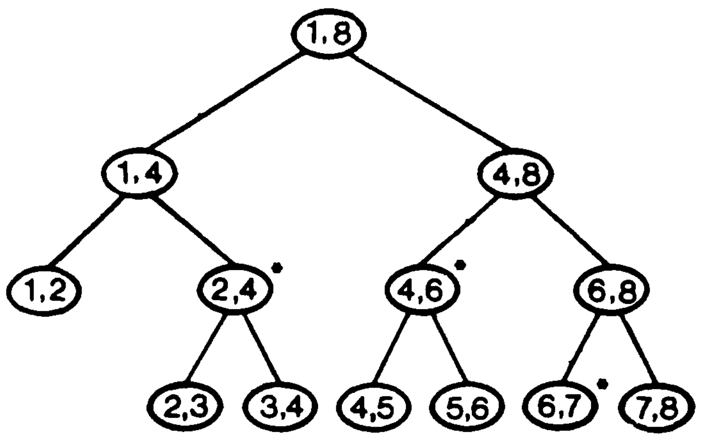


Figure 1: The segment tree $T(1,8)$ of the interval [1,8]
The starred nodes represent the interval [2,7] on $T[1,8]$
Node (1,2) is empty, (4,5) is full and (1,8) is partial.

To insert (or delete) an interval segment, find the corresponding marked nodes and add (or delete) the segment from each node's interval list. Since every node's interval list is no longer then m, insertion or deletion can be done in $O(t \log_2 k)$ time if t is the time required to insert (or delete) an element from an interval list of size $\leq m$. In general $t = \log_2 m$ if a balanced binary search tree is used when m is large. In some algorithms, t is a constant. For example, if one only needs to know how many segments marked a node, then the interval list is simply an integer value which is incremented or decremented. In this case the space needed to store T is linear.

Given $T(a, b)$ and an interval $[c, d] \subseteq [a, b]$, each node $v \in T(a, b)$ is classified as being (i) empty if $[c, d] \cap [L(v), R(v)]$ is empty, (ii) full if $[L(v), R(v)] \subseteq [c, d]$ and (iii) partial, if it is neither empty nor full. See Figure

1. Given a set of intervals, a node v is empty (or full) if and only if it is empty (or full) with respect to all the intervals.

All the line sweep algorithms in the following sections use the basic segment tree or some variation of it to organize the necessary information, for easy insertion and deletion of appropriate interval segments.

3. Boundaries and Perimeters

The union of a set of n rectangles consists of one or more connected regions. Its contour consists of a collection of disjoint cycles, composed alternately of vertical and horizontal edges, which specifies the outer boundaries as well as the holes, if any, of the region. See Figure 2. The algorithms in this section can be used to find the boundaries and perimeter of the region from its medial axis transform.

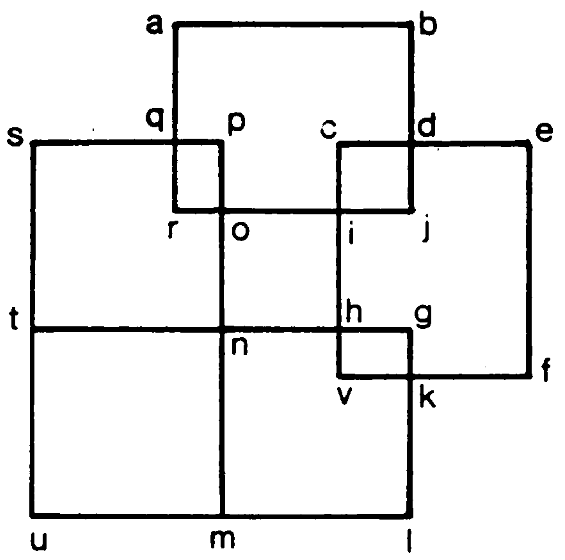


Figure 2: The contour is the set of two cycles: $\{abdefklmutsqa\ ,\ onhio\}$.

The algorithm in [4] determines the contour in two phases: the first finds all the vertical edges and the second links them with horizontal edges. The first phase uses a vertical scan line which sweeps from left to right. The horizontal edges of the rectangles divide the vertical scan line into a number of interval segments. If I is the set of such vertical intervals just before (or after) some vertical left (or right) edge E of a rectangle, then the contribution of E to the contour of the region is $I' \cap E$ where I' is the complement of I. For example, in Figure 2, just before edge ar, $I = \{st, tu\}$ and $I' \cap \{ar\} = \{aq\}$; just after edge pm, $I = \{ar, mn\}$ and $I' \cap \{pm\} = \{on\}$. A segment tree is used to determine $I' \cap E$. First, the y-coordinates of the horizontal edges are sorted and mapped onto $\{1, \ldots, m\}$ $(m \leq 2n)$ where each horizontal edge (determined by its y-coordinate) is associated with its position in the sorted list. A segment tree $T(1,m)$ is built. When the vertical scan line sweeps from left to right, and a left edge is encountered, the y-coordinates of its two endpoints are mapped into $\{1, \ldots, m\}$ and the segment is inserted into $T(1,m)$. Similarly, a right edge is deleted from $T(1,m)$. Note that at each node in T, only a count of the number of times it was marked needs to be maintained. This value is increased or decreased as the node is marked or unmarked. The union of the nodes with nonzero counts represents the interval I. For each node v in T, if $CONTR(v)$ denotes $[L(v), R(v)] \cap I'$, then

$$CONTR(v) = \begin{cases} \emptyset \text{ if } v \text{ is full or there is an ancestor } u \text{ of } v \text{ such that } u \text{ is full} \\ [L(v), R(v)] \text{ if } v \text{ is empty} \\ CONTR(\text{LEFTCHILD}(v)) \cup CONTR(\text{RIGHTCHILD}(v)) \\ \quad \text{if } v \text{ is partial.} \end{cases}$$

Therefore $E \cap I' = \bigcup_{v \in S} CONTR(v)$ where S is the set of $O(\log_2 m)$ nodes representing the edge E in T. The edge(s) obtained from $E \cap I'$ in general are unions of segments. They must be processed, i.e., contiguous intervals must be collated.

Once the vertical edges are found, the horizontal edges can easily be determined. Basically the set of endpoints of the vertical edges is sorted in ascending order of the y-coordinates, and then in ascending order of the x-coordinates if two points have the same y-coordinate. In the sorted list (x_1,y_1), (x_2,y_2),(x_3,y_3), $\cdots$, $y_{2i-1} = y_{2i}$ for $i \geq 1$. Then (x_{2i-1}, y_{2i-1}) and (x_{2i}, y_{2i}) are the endpoints of a horizontal edge.

We can assign directions to the edges consistently so that we can determine if a cycle represents the outer boundary or a hole by observing the direction at some ext'eme (say, south-west) corner of a cycle. The direction of a vertical edge is "up" if it arises from a left edge of a rectangle, and "down" otherwise. This can easily be done when the edge is identified. In phase two, if at each endpoint (x, y) of a vertical edge, the direction d of the vertical edge and the y-coordinate y' of the other end of the vertical edge are also recorded, then the horizontal edge $\{(x_{2i-1},y_{2i-1})(x_{2i},y_{2i})\}$, i.e., $y_{2i-1} = y_{2i}$, has direction (from left to) right if d_{2i-1} is down and $y_{2i-1} < y'_{2i-1}$, **or** if d_{2i-1} is up and $y_{2i-1} < y'_{2i-1}$. Otherwise the horizontal edge has direction (from right to) left.

If the contour has p edges, the above algorithm uses $O(n \log n + p \log \frac{2n^2}{p})$ time and $O(n + p)$ space.

In [5], an additional data structure called the contracted segment tree is used to improve the time complexity to $O(n \log n + p)$. To find the vertical edges, one needs to compute the contribution of rectangle edge e. First the segment tree nodes which represent the y-interval of e are located. But if some of these nodes or their ancestors are full with respect to the segments already in the tree, their contribution to $E \bigcap I'$ is empty. Thus we need to find only those parts of the segment tree which are "free" with respect to the segment being added. This is done in [4] by traversing the subtree of the relevant nodes, and not reporting the FULL nodes. In [5], the "free" parts in the subtree of a node are attached to the node as another segment tree, the contracted segment tree, which stores only the "gaps" in the subtee rooted at that node. This allows us to report the gaps in $O(\alpha)$ time where α is the number of gaps in the subtee at the given node, and thus yields an optimal time algorithm.

Clearly, the contour algorithm can be easily modified and simplified to find the perimeter which equals the total length of the borders of the regions. It also gives us the number of connected components in the union (just count the number of outer borders).

7. Conversion between representations

A subset can be specified using various representations other than a union of rectangles. For example, it can be specified by the boundaries of its regions, by its run length code, or by its quadtree [2]. This section discusses conversion of the union-of-rectangles representation to and from the boundary and run length representations. Given a quadtree, its set of (black) leaf nodes is a (non-minimal) union-of-rectangles representation. Conversely, given even a single rectangle, depending on its position, the corresponding quadtree can have O(image diameter) leaves. We will not discuss conversion between quadtrees and unions of rectangles in detail here.

7.1. Boundaries

The algorithm in Section 3 finds the boundaries of a subset from its representation by a set of rectangles. The outer boundaries, given by the (x, y) coordinates of the vertices, are specified in one direction (clockwise), and the holes are specified in the opposite direction (counterclockwise).

Suppose the corners (vertices) of the contours (boundaries) of a subset are given using the above convention. We first consider the following simple algorithm to find a set of rectangles whose union is the subset.

1. The vertices of the boundary are sorted in increasing order. Let $u_1 < u_2 < \cdots < u_m$ be the distinct x-coordinates. We will use a vertical sweepline which stops at each u_s $(1 \leq s < m)$.

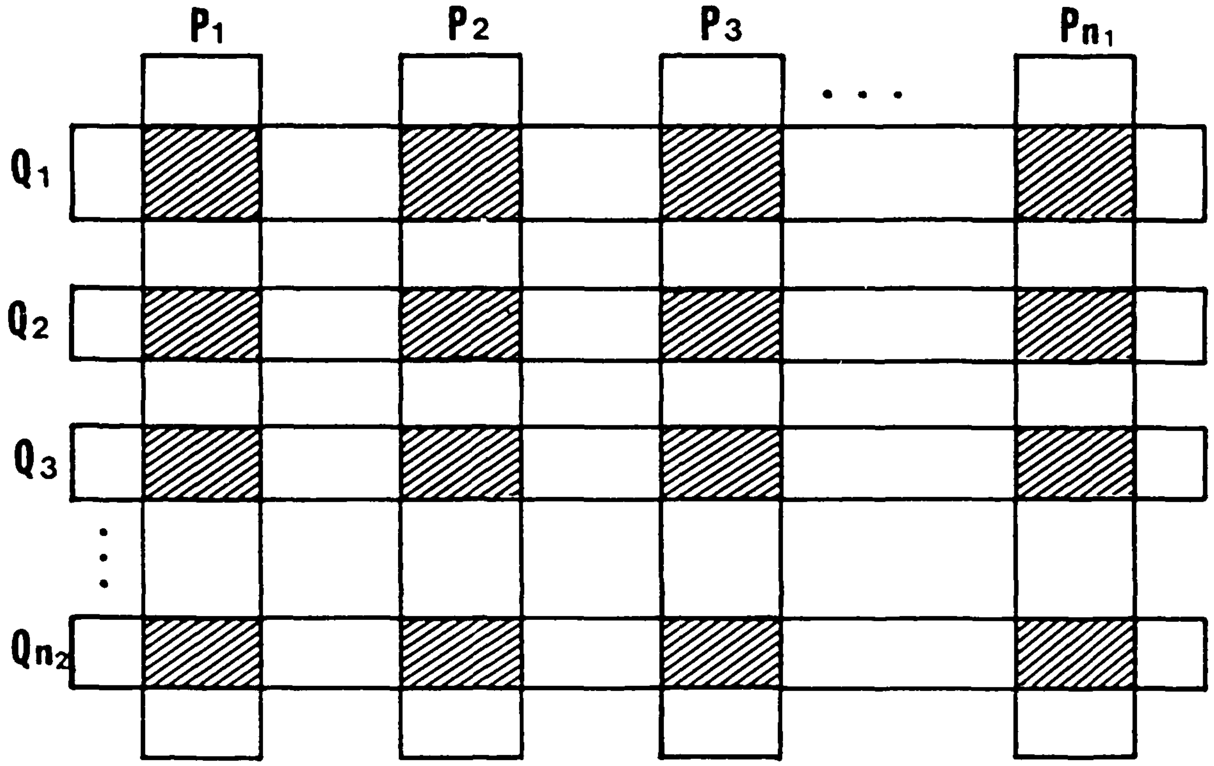


Figure 8: Regions $S_1 = \bigcup_{i=1}^{n_1} P_i$, $S_2 = \bigcup_{j=1}^{n_2} Q_j$. The P_i's are disjoint, the Q_j's are disjoint, and $S_1 \bigcap S_2$ is the union of $n_1 n_2$ disjoint rectangles.

rectangles whose union is the complement. In fact, the sorting step in the algorithm need not be performed since it can be done in stage 1. The time complexity of stage 2 is thus $O(p \log h)$ where h is the number of horizontal edges a vertical sweep line can cross. The complement can be found in $O(n \log n + p \log h)$ time, where $p \leq n^2$ and $h \leq n$.

by intersecting each P_i in S_1 with each Q_j in S_2. This takes $O(n_1 n_2)$ time and the intersection has a maximum of $n_1 n_2$ rectangles. Figure 8 shows the case where $S_1 \cap S_2$ has $n_1 n_2$ disjoint rectangles, and each P_i intersects every Q_j; in this case it takes $O(n_1 n_2)$ time to find the intersection. One can improve the efficiency by first sorting the rectangle vertices in increasing (x, y) order: sorted list L_1 for the P_i's and sorted list L_2 for the Q_j's. Now one can go down list L_1, and for each rectangle P_i, one only needs to intersect it with those Q_j which fall in the range of P_i. Of course, in the worst case (as the example in Figure 8 shows), this still takes $O(n_1 n_2)$ time. In fact any algorithm would need to take at least $O(n_1 n_2)$ time for the sets shown in Figure 8.

We find that the segment tree is not particularly useful for this problem. In order to produce the rectangles in the intersection, one needs to know not only the active segments at each stopping position of the scan line, but also the x-coordinate of the left edge which makes the segment active. Moreover, to search for the Q_j's which intersect a vertical edge of P_i may entail searching the entire subtree of the node corresponding to that vertical segment. Again, the sets shown in Figure 8 would cause this to happen.

The complement of a set of n rectangles with respect to an enclosing outer rectangle can be found in two stages. First we can use the $O(n \log n + p)$ algorithm in Section 4 to find the contour of the region; p is the number of edges in the contour. The contour of the complement is this contour with all edge directions reversed, together with the outer rectangle. Then we can use the boundary to union-of-rectangles conversion algorithm in Section 7.1 to get a set of

6. MATs of derived sets

The algorithms in the previous sections show that geometric properties can be computed from MATs quite efficiently. This section discusses algorithms for obtaining MAT representations for subsets derived from given subsets by set-theoretic operations such as union, intersection, complement and windowing, where the given subsets are represented by MATs. Since it is well-known that the problem of finding minimal rectangle covers for polygons is NP-hard [10], the problem of minimizing the numbers of rectangles in the MAT representations of the derived sets is not discussed.

First we consider the problem of windowing. Given a region represented by a set of m upright rectangles and a rectangular window, it is easy to find that part of the region which is inside the window. Since the intersection of two upright rectangles is either empty or another upright rectangle, one can find the intersection of ιne window with each of the n given rectangles and report all of the non-empty intersections. Thus windowing is a linear-time operation.

Given two sets of rectangles, each representing a region, the set union of these two sets represents the union of the two regions. Of course, the resulting set of rectangles is in general not minimal. Note that in the case where the two given regions are disjoint, the union of the sets of ·ectangles is the best one can do.

The intersection of two regions $S_1 = \bigcup_{i=1}^{n_1} P_i$ and $S_2 = \bigcup_{j=1}^{n_2} Q_j$ where P_i Q_j are upright rectangles, $S_1 \bigcap S_2 = \bigcup_{i=1}^{n_1} \bigcup_{j=1}^{n_2} (P_i \bigcap Q_j)$, can be found

$$\iint_A x^k \, dx \, dy$$

$$= \int_{u_1}^{u_2} \int_{v_3}^{v_7} x^k \, dx \, dy + \int_{u_2}^{u_3} \int_{v_3}^{v_8} x^k \, dx \, dy + \int_{u_3}^{u_4} \int_{v_1}^{v_8} x^k \, dx \, dy + \int_{u_4}^{u_5} \left[\int_{v_1}^{v_4} x^k \, dx \, dy + \int_{v_5}^{v_8} x^k \, dx \, dy \right]$$

$$+ \int_{u_5}^{u_6} \int_{v_1}^{v_8} x^k \, dx \, dy + \int_{u_6}^{u_7} \int_{v_1}^{v_6} x^k \, dx \, dy + \int_{u_7}^{u_8} \int_{v_2}^{v_6} x^k \, dx \, dy$$

$$= \frac{u_2^{k+1} - u_1^{k+1}}{k+1}(v_7 - v_3) + \frac{u_3^{k+1} - u_2^{k+1}}{k+1}(v_8 - v_3) + \frac{u_4^{k+1} - u_3^{k+1}}{k+1}(v_8 - v_1)$$

$$+ \frac{u_5^{k+1} - u_4^{k+1}}{k+1}\left[(v_4 - v_1) + (v_8 - v_5)\right] + \frac{u_6^{k+1} - u_5^{k+1}}{k+1}(v_8 - v_1)$$

$$+ \frac{u_7^{k+1} - u_6^{k+1}}{k+1}(v_6 - v_1) + \frac{u_8^{k+1} - u_7^{k+1}}{k+1}(v_6 - v_2).$$

$M = \sum_{1 \le i \le 8} \frac{u_{i+1}^{k+1} - u_i^{k+1}}{k+1} M(i)$, where $M(i)$ is the one-dimensional measure of the active region between u_i and u_{i+1}. Since $M(i)$ is obtained as before and the u_i are known, we can evaluate the required integral. The integral $\iint_A y^k \, dx \, dy$ can be evaluated by rotating the figure by 90° and evaluating $\iint_A x^k \, dx \, dy$. The integral $\iint_A x \, dx \, dy$ can be evaluated concurrently with $\iint dx \, dy$, and thus the moments can be obtained in the same time complexity as the area.

known, and the $M(i)$ are evaluated as described above. Thus, $\iint\limits_A x \, dx \, dy$ can be determined in time $O(n \log n)$. $\iint\limits_A dx \, dy$ is the area of the region. $\bar{y}$ may be obtained in the same way, rotating the axes by $90°$.

The moment of inertia about the origin is given by

$$\iint\limits_A (x^2 + y^2) \, dx \, dy = \iint\limits_A x^2 \, dx \, dy + \iint\limits_A y^2 \, dx \, dy.$$

In general, the problem of evaluating the moments is one of evaluating integrals of the form $\iint\limits_A x^k \, dx \, dy$ and $\iint\limits_A y^k \, dx \, dy$. For example, if A is the region defined by the union of the rectangles in Figure 7, then

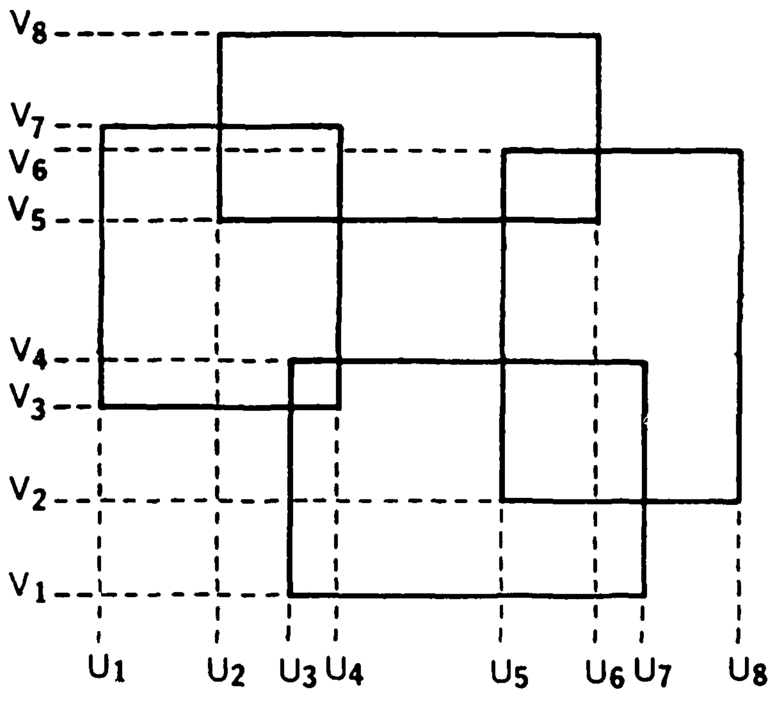


Figure 7. Illustration of moment computation.

When inserting an interval segment into the tree, instead of simply marking the nodes as in Section 2, we mark all the nodes of the 1-umbrella of that segment. At each node of the segment tree, the following two values are maintained: (1) a count of the number of times it figures as a full node of some umbrella, and (2) the total length of the fragments in its subtree which are covered by 1-umbrellas through it or below it. The value of the second field at the root gives M_i at scan position u_i. Deletion of a segment updates the above values too. We need the count field (1) because a given node can belong to several umbrellas. Since each partial sum can be found in $O(\log n)$ time, the area can be determined in $O(n \log n)$ time and the segment tree only needs $O(n)$ space for n rectangles.

In the rest of this section, we use the segment tree with 1-umbrellas to determine the moments of the region.

The centroid (or center of gravity) $(\bar{x}, \bar{y})$ of a region A is given by

$$\bar{x} = \frac{\iint_A x\, P(x, y)\, dx\, dy}{\iint_A P(x, y)\, dx\, dy} \ , \quad \bar{y} = \frac{\iint_A y\, P(x, y)\, dx\, dy}{\iint_A P(x, y)\, dx\, dy}$$

where $P(x, y)$ is the density at point (x, y). If we assume uniform density throughout, then

$$\bar{x} = \frac{\iint_A x\, dx\, dy}{\iint_A dx\, dy} \ , \quad \bar{y} = \frac{\iint_A y\, dx\, dy}{\iint_A dx\, dy}$$

We can find these incrementally at each stop position u_i of the vertical scan line by observing that $\iint_A x\, dx\, dy = \sum_{1 \le i \le k-1} \frac{1}{2}(u_{i+1}^2 - u_i^2) M(i)$. The u_i are tained in the interval t represents, and q is not contained in t's leftchild or rightchild. In other words, $[v_i, v_j]$ is split at t where the beginning part of it goes to t's left subtree and the rest goes to t's right subtree. Let Ltip, Rtip be the leftmost (rightmost) marked node for q in the left (right) subtree of t, i.e., $L(\text{Ltip})=v_i$, $R(\text{Rtip})=v_j$. Ltip, Rtip are full with respect to q.

The 1-umbrella of a segment $[v_i, v_j]$ consists of the node t as defined above, all nodes along the path from t to Ltip and their rightchildren (if any), and all nodes along the path from t to Rtip together with their leftchildren (if any). See Figure 6. Since an 1-umbrella has at most four nodes at each level, it has at most $O(\log n)$ nodes for any segment and it can be built in $O(\log n)$ steps.

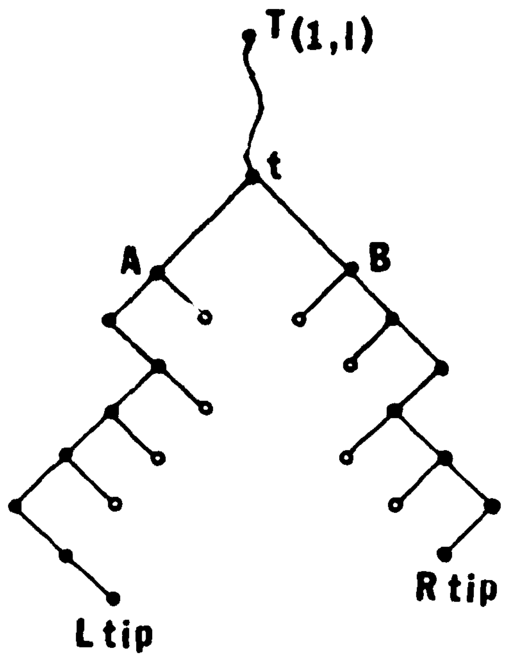


Figure 6. A 1-umbrella of a segment $[v_i, v_j]$. Here $[v_i, v_j] \subseteq [L(t), R(t)]$ but each child of t contains a portion of $[v_i, v_j]$.

5. Area and moments

The area (or measure) of a set of m rectilinear rectangles is the area covered by (= the number of pixels in) their union. The area together with the perimeter (see Section 3) gives us information about the compactness of the region. The measure problem was first solved in 1 and 2 dimensions in [3], and a generalized solution in d dimensions was presented in [9].

The algorithm to find the area in [9] uses the sweep line method and a version of the segment tree. First the x-coordinates of the vertical edges are sorted to get the list $u_1, u_2, \ldots, u_k$ $(k \leq 2n)$. The vertical scan line will be positioned at each of the u_i's. Let M_i be the length of the active interval segments when the scan line is at u_i; then the area of the region $M = \sum_{1 \leq i < k} M_i (u_{i+1} - u_i)$. Thus, to determine the total area, we need to accumulate the partial areas. For this we need to determine the length of the active segments at each stopping position of the scan line. The algorithm will calculate M_{i+1} by applying a correction to M_i.

The y-coordinates of the horizontal lines are sorted, denoted by $\{v_1, v_2, \ldots, v_l\}$, $l \leq 2n$. The segment tree $T(1, l)$ is built. At each scan position u_i, $1 \leq i \leq k$, the active segments are marked on the segment tree together with some information described below.

Let $q = [v_i, v_j]$, $v_i < v_j$ be a vertical interval segment. In finding and marking the tree nodes representing $[v_i, v_j]$ (see Section 2), starting from the root node, let t be the node in $T(1, l)$ such that $[v_i, v_j] \subseteq [L(t), R(t)]$, i.e., q is con-

Once we obtain the list of all such pairs, we collect pairs which have at least one component in common, to form the connected components.

This algorithm presented in [6] determines the connected components of n rectilinear rectangles in $O(n \log n)$ time and $O(n)$ space.

Instead of using a priority search tree as in [5], we can use a segment tree to keep track of the active intervals. Specifically, the y-coordinates of the horizontal edges are sorted and mapped to $\{1, \ldots, k\}(k \leq 2n)$. A segment tree $T(1,k)$ is built. At each node we keep a count of the interval segments which marked the node and one of its descendants (subintervals). This is similar to the tree used in Section 3 except that here the count represents partial instead of full nodes. When a leftside (rightside) vertical segment is inserted (deleted) that count of all the nodes along the path from the root to the marked nodes is increased by 1. Thus insertion and deletion can be done in $O(\log n)$ time. To test if a given interval intersects any of the intervals in the segment tree, we locate the tree nodes v representing the test interval, i.e., $[L(v), R(v)] \subseteq$ test interval. If any of the nodes has a count >0, then it has a subinterval that intersects the test interval. Again this can be done in $O(\log n)$ time. Hence using the segment tree we can achieve the same time ($O(n \log n)$) and space ($O(n)$) complexity as the algorithm in [6].

As the vertical sweep line moves from left to right passing through the rectangles, both the priority search tree and the illuminator tree are updated as required. When a new rectangle A starts in the illuminator tree we may have to coalesce a collection of intervals lying between the endpoints. For each interval B being coalesced, if B intersects any of the current priority search tree intervals, we output the pair (A , B). For example, consider Figure 5.

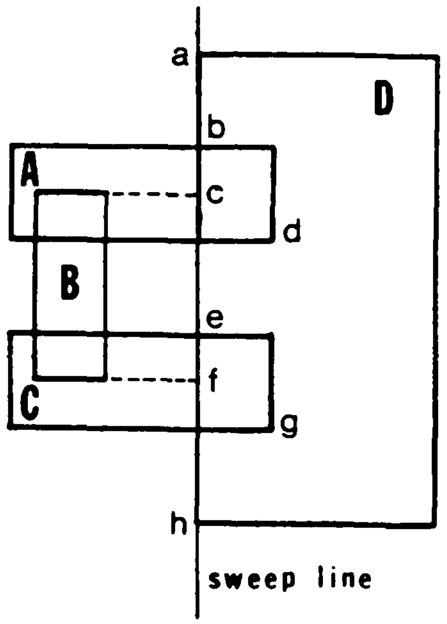


Figure 5. Example of tree updating.

Just before the sweep line reaches edge ah , the priority search tree contains intervals bd and eg while the illuminator tree has segments ab , bc , cf , fg , gh . On meeting ah , we have to merge intervals ab , bc , cf , fg and gh to obtain the new illuminator ah . Of the segments being coalesced, only bc , cf and fg share points with bd and eg . Thus, we report pairs (A , D), (B , D) and (C , D).

new rectangle starts in the illuminator tree, we must do the following: (1) Locate the two endpoints of the left side of the rectangle in the current set of intervals and split the intervals containing the endpoints. This can be done in $O(\log n)$ time. (2) Merge all t intervals lying between the two endpoints, because in these parts, the new rectangle becomes the closest rectangle to its left. This can be done in $O(t)$ time.

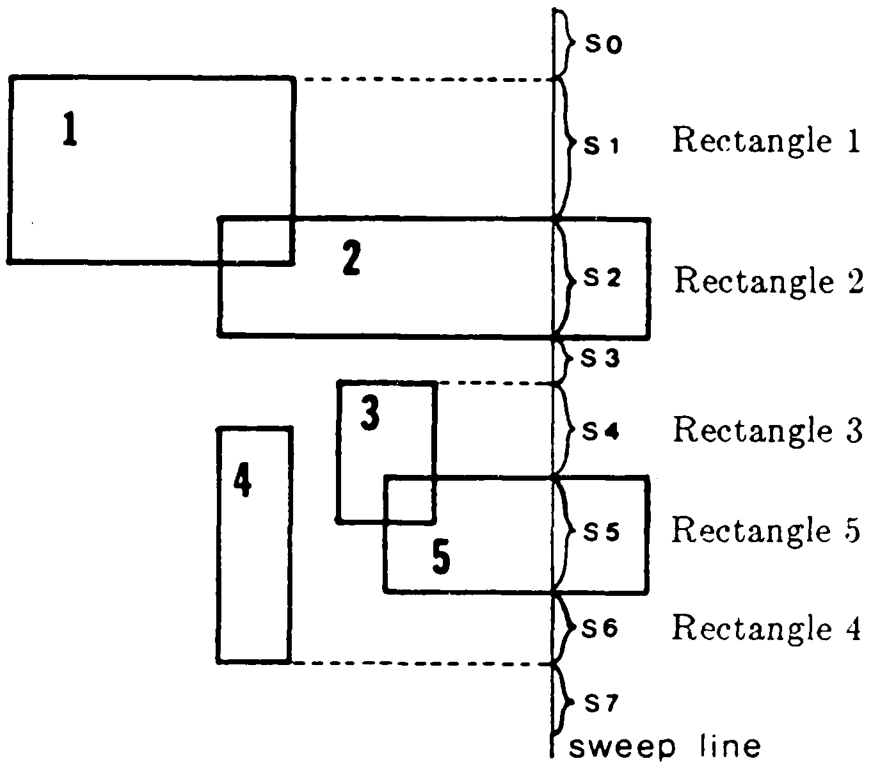


Figure 4. $S0, S1, \ldots, S6, S7$ are the segments in the illumination tree at the position of the sweep line. In parentheses are the rectangles that illuminate each of the segments.

"dynamic" priority search tree. The priority search tree uses $O(n)$ spaces to represent n intervals using a balancing scheme as is done in working with AVL trees [7]. Insertion and deletion of intervals can be done in $O(\log k)$ time, and we can decide if a given interval intersects any of the intervals represented on the tree in $O(\log k)$ time where $k \leq n$ is the number of nodes in the tree. In the algorithm to find the connected components, the priority search tree is used to represent the vertical intervals which are the intersections of the current sweep line with the active rectangles.

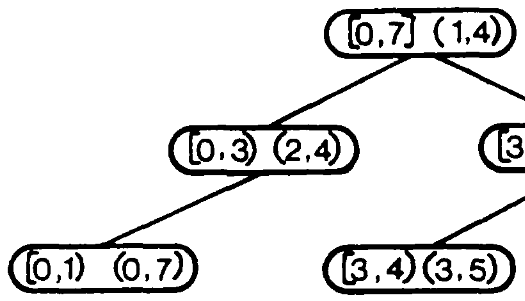


Figure 3. A priority search tree representing the intervals (1,4),(2,4),(4,7),(0,7),(3,5).

The illuminator tree represents the vertical intervals which form a partition of the sweep line, such that each interval is the projection of the left side of the nearest rectangle to the left of the sweep line. See Figure 4. The illuminator tree can be implemented as a balanced 2-3 tree [8] using $O(n)$ space. Each node in the tree represents, say, the bottom endpoint of the corresponding segment. When a rectangle ends, there is no change in the illuminator tree. But when a

4. Connected components

Given a set of n rectilinear rectangles, the connected components of the region covered by the rectangles can be determined in time $O(n \log n)$ and space $O(n)$ [6]. The connected components are specified by lists of rectangles where the rectangles in each list belong to the same connected region. The algorithm in [6] uses the line sweeping method and it works in two phases. The first phase produces a list of pairs of rectangles which belong to the same connected component. The second phase traverses the graph defined this pair list to obtain a list of rectangles for each component.

The first phase of the algorithm uses two data structures, the priority search tree [7] and the illuminator tree.

A priority search tree $T(a, b, c)$ can be defined as a binary tree such that T has a root v with $L(v)=a$, $R(v)=b$ and $P(v)=c$; if $|R(v)-L(v)|>1$, v has a leftson $T(L(v), \lfloor\frac{L(v)+R(v)}{2}\rfloor, d)$ where $c \leq d$; and v has a rightson $T(\lfloor\frac{L(v)+R(v)}{2}\rfloor, R(v), d')$ where $c \leq d'$. Thus the $T(L(\text{root}), R(\text{root}))$ is a segment tree as defined in Section 2, and the $P(v)$'s satisfy the properties of a priority queue. This priority search tree represents a set of n intervals $(s_1, e_1), \ldots, (s_n, e_n)$ where $s_i < e_i$, and the s_i's are distinct. The values of the s_i's are used to build a segment tree $T(a, b)$ (a = minimum of the s_i's, b = maximum of the s_i's $+1$), and each interval is associated with a tree node v such that $L(v) \leq s_i < R(v)$ and $P(v)=e_i$. See Figure 3. [6] also discusses how the constraints of distinct s values may be removed and how to create a

2. Initialize a list L to be empty. In general, L is an ordered list $(l_1, l_2, \ldots, l_d)$ in increasing order. It represents the y-coordinates of all the horizontal edges which intersect the vertical sweepline.

```
3. For $S := 1$ to $m$ do
   begin
      let $y_1 < y_2 < \cdots < y_k$ be the coordinates of the vertices
      with $x$-coordinate $u_s$;
      for $i := 1$ to $k$ do
         if $y_i$ is already in $L$, i.e., $l_j = y_i$ for some $j$
         then delete $l_j$ from $L$, since this signifies the righthand end
         of a horizontal edge
         else insert $y_i$ into $L$;
      after all the $y_i$'s are properly inserted, $L = (l_1, \ldots, l_{2t})$;
      for $i := 1$ to $2t$ by step 2
         Output rectangle $[l_i, l_{i+1}] \times [u_s, u_{s+1}]$, i.e., rectangles with vertical
         edges $l_i$ to $l_{i+1}$ and horizontal edges $u_s$ to $u_{s+1}$;
   end.
```

The above algorithm uses a vertical sweep line going from left to right. The sweep line stops at each vertical edge, finds all the horizontal edges it crosses and outputs rectangles of width = next stop position - current stop position. If there are n vertices on the contours, the time complexity of the algorithm is $O(n \log n + n \cdot h)$ where h = the maximum number of horizontal edges any vertical line crosses; h can be as large as n in the worst case. The number of rectangles produced is also $O(n \cdot h)$. Figure 9 shows that a long rectangle could be cut (unnecessarily) into many small rectangles by this algorithm since all rectangles output are of width $u_{i+1} - u_i$. Also at each u_i, the entire list L is examined.

The following algorithm uses the same basic principle as the one above, but an element of L is examined only if it is being deleted (the righthand end of a horizontal edge) or if some new horizontal edges are being inserted next to it. A

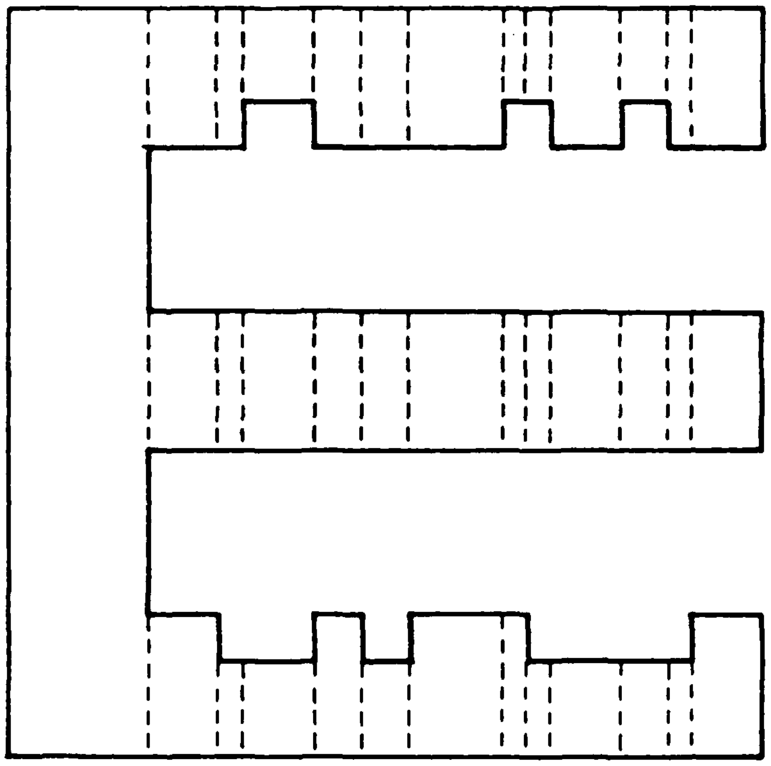


Figure 9: The long smooth horizontal rectangle in the middle of the E is cut into many small rectangles.

rectangle is output when it can no longer be extended to the right because of the presence of a corner or some other vertical edge. In this algorithm, the endpoints of a horizontal edge need to know the direction of the edge ($\rightarrow$ or $\leftarrow$, using the convention that clockwise is outer boundary, counterclockwise is a hole) and each l_i of L must also record the direction and a value x-tag which indicates the left-most position of the edge which has not been included in any of the rectangles output so far.

1. Each vertex $v = (x, y)$ of the boundary determines $dir(x, y)$, the direction of the horizontal edge at the vertex (its value is either $\rightarrow$ or $\leftarrow$).

2. The vertices are sorted in increasing (x, y) order. Let $u_1 < u_2 < \cdots < u_m$ be the distinct x-coordinates. For each u_s $(1 \leq s \leq m)$, let $y_{s1} < y_{s2} < \cdots < y_{sk_s}$ be the y-coordinates of the vertices with x-coordinates u_s.

3. L is a list $(l_1, l_2, \ldots, l_{last})$ such that $l_i = (Y\text{-val}, D, X\text{-tag})$. L is initialized to $\{(y_{11}, dir(u_1, y_{11}), u_1), \ldots, (y_{1k_1}, dir(u_1, y_{1k_1}), u_1)\}$.

4.
```
For $S := 2$ to $m$ do
    (* At each sweepline position, examine each corner of the boundary having this $x$ value *)
    for $t := 1$ to $k_s$ do
        if no element in $L$ has $Y$-val $= y_{s,t}$
        then (* the beginning of a new horizontal edge *)
            begin
            let $a = (y_{s,t}, dir(u_s, y_{s,t}), u_s)$.
            if $a$ is to be inserted between $l_i$ and $l_{i+1}$ for some
            $1 \leq i < last$
            then (* $a$ is not new first element or last element of $L$ *)
                (* look at $l_i$ and $l_{i+1}$ *)
                begin
                    if $l_i.D = \leftarrow$ and $l_{i+1}.D = \rightarrow$ and $l_i.X$-tag
                    $= l_{i+1}.X$-tag $< u_s$ then output $[l_i.Y$-val,
                    $l_{i+1}.Y$-val$] \times [l_i.X$-tag, $u_s]$,
                        and set $l_i.X$-tag and $l_{i+1}.X$-tag to $u_s$.
                end;
            insert $a$ in $L$
        end
    else (* an element $l_j$ in $L$ has $Y$-val $= y_{s,t}$, i.e., the end of a hor-
    izontal edge in $L$ *)
        begin
            suppose $l_j.Y$-val $= y_{s,t}$; then we need to look at
            $l_{j-1}$ or $l_{j+1}$ to determine the rectangle to output.
            if $dir(u_s, y_{st}) = \rightarrow$
            then output $[l_{j-1}.Y$-val, $y_{st}] \times [l_j.X$-tag, $u_s]$
                and set $l_{j-1}.X$-tag to $u_s$
            else (* $dir(u_s, y_{st}) = \leftarrow$ *)
                output $[y_{st}, l_{j+1}.Y$-val$] \times [l_j.X$-tag, $u_s]$
            **and** set $l_{j+1}.X$-tag to $u_s$;
        delete $l_j$ from $L$.
    **end**
```

Using this algorithm, the middle rectangle of the E in Figure 9 will not be cut up into small pieces. The number of rectangles reported is $O(n)$ where n is the number of corners in the boundary, since every rectangle has at least one boundary vertex on one of its edges. The time complexity of this algorithm is $O(n \log n + nt)$ where t is the time for inserting or deleting an element of L. If L is implemented as a balanced binary search tree, insertion and deletion take $O(\log h)$ where $h \leq n$ is the maximum number of edges a vertical sweepline can cross. We can also doubly link the elements of L in increasing order to allow easy access to the immediate neighbors. Therefore the time complexity of this algorithm is $O(n \log n)$ since $h \leq n$.

7.2. Run length codes

Given a set of rectangles we can obtain its run length code by reporting the active segments on each row as runs of 1s and the inactive segments as runs of 0s. We assume that the vertices of the n rectangles are sorted on the key (y, x). We also assume that the rectangles are all contained within a bounding frame starting at x-coordinate M and extending up to x-coordinate N. The vertical dimension of the frame may be taken to be the vertical span of the set of rectangles itself. We pass a horizontal sweep line from top to bottom.

Let L be a sorted list of nodes, each representing a horizontal segment. Each node has fields LEFT and RIGHT for the left and right ends of the segments, and a COUNT field to represent the number of times a given portion of the x-axis was covered by segments. LINK is the pointer to the next node on L.

Let $y_1 > y_2 > \cdots y_k$ be the list of distinct y-coordinates.

```
for $i$ :=1 to $k$ do :
    begin
        let $l_1, l_2, \ldots, l_j$ be the horizontal segments with $y$-coordinate $y_i$.
    for $a$ :=1 to $j$ do :
        begin
            if $l_a$ is the top end of some rectangle then
                begin COUNT1 ← 0; $P$ ← start of $L$;
                    output a run of 0s of length LEFT($P$) - $M$ + 1;
                    repeat
                        COUNT1 ← COUNT1 + RIGHT($P$) - LEFT($P$) +
                        1;
                        if RIGHT($P$) = LEFT(LINK($P$)) then    (* continue
                        run *)
                          $P$ ← LINK ($P$)
                        else begin output a run of 1s of length COUNT1
                                if LINK($P$) = nil then output a run of 0s
                                of length $N$ - RIGHT($P$) + 1
                                else output a run of 0s of length
                LEFT(LINK($P$) - RIGHT($P$) + 1;
                $P \leftarrow$ LINK($P$); COUNT1 $\leftarrow 0$
            end
    until the proper place is found for $l_a$ on the sorted list $L$
        (i.e., $l_a$ lies between two nodes of $L$).
    One of the following cases must hold:
    Case 1: $l_a$ is covered entirely by some segments on $L$:
                Split those segments into parts covering $l_a$ and
                parts not covering $l_a$. Increment the COUNT
                field of the first nodes by 1.
    Case 2: $l_a$ is partially covered: Split the segments into
                parts covering $l_a$ and parts not covering $l_a$. Insert
                that part of $l_a$ not covered, into $L$ and increment
                the COUNT fields of the covered parts.
    Case 3: $l_a$ is not covered: Insert $l_a$ with a COUNT of 1.
        end (* if $l_a$ is top end *)
    else (* $l_a$ is a bottom end *)
        begin
                Scan the list $L$ as before, in the if part, output-
                ting runs of 0s and 1s, but decrement the
                COUNT fields of all segments contained in $l_a$ by
                1. Delete those segments whose COUNT becomes
                0.
            end
        end (* for each $a$ *)
    end (* for each $i$ *)
```

In this way, we get the run length code of all those rows y_i, $1 \leq i \leq k$ on which a horizontal line of some rectangle is incident. There are $O(n)$ stopping points of the horizontal sweep, and the list L can contain $O(n)$ nodes, because there can be $O(n)$ distinct vertical coordinates. The time complexity is $O(n^2)$. The run length code of any other row is the same as that of the preceding row.

To convert run length code representation to union-of-rectangles, we maintain a list L of nodes representing rectangles. Each node has a "rowspread" field and a "columnspread" field. The rowspread field indicates that the rectangle specified by that node spans the two rows specified in this field. The columnspread field indicates that the rectangle specified by that node spans the two columns specified in this field.

Initially, L is empty.

```
For each row of the run length code do:
    Let the run be $a_1$ 0s, $a_2$ 1s, $a_3$ 0s, $a_4$ 1s, . . . . $a_{2n-1}$ 0s, $a_{2n}$ 1s,
    $a_{2n+1}$ 0s.
    Let current position on list $L$ be the start of list $L$
    For each run of 1s in this row do:
        Scan through $L$ starting from the current position until a node $l$ is
        found whose columnspread intersects the columnspread of the current
        run of 1s. Output rectangles corresponding to all nodes before $l$.
```

Let columnspread $(l) = c_1, c_2$.

Let columnspread of current run of 1s $= d_1, d_2$.

case 1: $c_1 = d_1, c_2 = d_2$: increase rowspread of l

case 2: $c_1 = d_1, d_1 > d_2$: split l by increasing its rowspread by 1 and making the columnspread run till d_2. Output a rectangle with rowspread equal to that of l previously and columnspread from d_2 to c_2.

case 3: $c_1 = c_2, d_2 > d_1$: increment rowspread of l and append a new node after l, whose columnspread is from d_1 to the end of the current run of 1s.

case 4: $c_1 < c_2, d_1 = d_2$: split l by incrementing its rowspread, and changing its columnspread to start from d_1. Output a rectangle with rowspread the same as that of l and columnspread from c_1 to d_1.

case 5: $c_1 < d_1 < c_2 < d_2$: split l by incrementing its rowspread, and changing its columnspread to $d_1 - c_2$. Output a rectangle with the old rowspread of l and columnspread from c_1 to d_1. Append a node after l with columnspread from c_2 to d_2.

case 6: $d_1 < c_1 < d_2 < c_2$: split l by incrementing the rowspread, and changing its columnspread to $c_1 - d_2$. Output a rectangle with old rowspread of l and columnspread from d_2 to c_2. Append a node before l with columnspread d_1 to c_1.

case 7: $c_1 < d_1 < d_2 < c_2$: split l by incrementing its rowspread, and changing the columnspread to $d_1 - d_2$. Output two rectangles with columnspread equal to the previous value of l and rowspread from c_1 to d_1, and d_2 to c_2.

case 8: $d_1 < c_1 < c_2 < d_2$: split l by incrementing its rowspread, and appending two nodes, one before and one after l, with columnspreads d_1 to c_1, and c_2 to d_2.

The rectangles we output may be degenerate ones, i.e., a single point or a single line. This may be avoided by increasing the resolution so that single pixels become rectangles with non-empty interior. The number of rectangles is bounded by the number of corners in the contour of the region, which is $\leq$ the number of runs of 1s. In the worst case, the time complexity is O(image size) when the image is a checkboard.

8. Concluding Remarks

The algorithms in this note all use the line sweep method and the segment tree or its variants as the supporting data structure. They are time optimal algorithms: $O(n \log n + p)$ for the contour of a union of n rectilinear rectangles, where p is the number of contour pieces; $O(n \log n)$ for the connected components, area, centroid and moments.

These time and space efficient algorithms can be useful in image processing because the medial axis transform of a region is a set of rectilinear squares. They show that geometric properties can be obtained from the MAT quite efficiently. Moreover, often regions can be covered by a lot fewer rectangles [10] than the number of squares in a MAT. Thus a set of rectangles is a useful compact representation of regions.

The problems we considered in Sections 3–5 can also be solved using divide and conquer methods [11]. These algorithms divide the plane into frames which are vertical strips between two vertical lines (as defined by the vertical edges of rectangles). Then the problem is solved for each of the frames. Some information which allows the merging of the solutions is also calculated. Finally the solutions for the subproblems are merged. The performance of the divide and conquer algorithms matches those of the line sweeping method.

A set of rectilinear rectangles can be obtained from other representations of regions and from performing set-theoretic operations on given MATs.

References

[1] H. Blum, "A transformation for extracting new descriptors of shape", *Models for Perception of Speech and Visual Form,* W. Wathen-Dunn, Ed., Cambridge, MA: MIT Press, 1967, 362-380.

[2] A. Rosenfeld and A.C. Kak, *Digital Picture Processing,* Second Edition, Vol. 2, Chapter 11, NY: Academic Press, 1980.

[3] J.L. Bentley, Solutions to Klee's rectangle probelms, Department of Computer Science, Carnegie-Mellon University, 1977.

[4] W. Lipski and F.P. Preparata, "Finding the contour of a union of iso-oriented rectangles", *J. Algorithms* **1**, 1980, 235-246 and *J. Algorithms* **3**, 1982, 301-302.

[5] R.H. Güting, "An optimal contour algorithm for iso-oriented rectangles", *J. Algorithms* **5**, 1984, 303-326.

[6] L. Guibas and J. Saxe, "Solution to <Problem 80-15>", *J. Algorithms* **4**, 1983, 177-181.

[7] E.M. McCreight, "Priority Search Trees", Xerox PARC Research Report CSL-81-5, 1982.

[8] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Reading, MA: Addison Wesley, 1974.

[9] J. van Leeuwen and D. Wood, "The measure probelm for rectangular ranges in d-space", *J. Algorithms* **2**, 1981, 282-300.

[10] D.S. Franzblau and D.J. Kleitman, "An algorithm for constructing regions with rectangles", *Proc. Sixteenth ACM STOC,* 1984, 167-174.

[11] R.H. Güting, "Optimal divide and conquer to compute measure and contour for a set of iso-rectangles", *Acta Informatica* **21**, 1984, 271-291.

[12] N. Ahuja and W. Hoff, "Augmented medial axis transform", *Proc. Workshop on Computer Vision: Representation and Control,* 1984, 251-256.

:nclassified
RITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE AD-A158934

EPORT SECURITY CLASSIFICATION :nclassified		1b. RESTRICTIVE MARKINGS N/A	
ECURITY CLASSIFICATION AUTHORITY N/A)ECLASSIFICATION/DOWNGRADING SCHEDULE N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
RFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-122 CS-TR-1497		5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A	
IAME OF PERFORMING ORGANIZATION University of Maryland	6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research	
ZIP Code) Center for Automation Research College Park, MD 20742		7b. ADDRESS (City, State and ZIP Code) Bolling Air Force Base Washington, DC 20332	
AME OF FUNDING/SPONSORING NIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-83-C-0082	
DDRESS (City, State and ZIP Code)		10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.	
TITLE (Include Security Classification) mputation of geometric properties from the dial axis transform in O(n logn) time			
PERSONAL AUTHOR(S) gela Y. Wu, S. K. Bhaskar, Azriel Rosenfeld			
nnical	13b. TIME COVERED N/A FROM ______ TO ______	14. DATE OF REPORT (Yr., Mo., Day) June 1985	15. PAGE COUNT 37
TATION			

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
IELD	GROUP	SUB. GR.	

ABSTRACT (Continue on reverse if necessary and identify by block number)

The digital medial axis transform (MAT) represents an image subset S as the union f maximal upright squares contained in S. Brute-force algorithms for computing geometric roperties of S from its MAT require time O (n^2), where n is the number of squares. Over he past few years, however, algorithms have been developed that compute properties for a nion of upright retangles in time O (n logn), which makes the use of the MAT much more ttractive. We review these algorithms and also present efficient algorithms for computing nion-of-retangle representations of derived sets (union, intersection, complement) and or conversion between the union of rectangles and other representations of a subset.

DISTRIBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL

FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE